```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

df=pd.read_csv('/content/car_evaluation.csv')
```

```
df.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

```
df.shape
```

```
(1727, 7)
```

```
col_names = ['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety', 'class']

df.columns = col_names
col_names
```

```
['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1727 non-null   object
 1   meant     1727 non-null   object
 2   doors     1727 non-null   object
 3   persons   1727 non-null   object
 4   lug_boot  1727 non-null   object
 5   safety    1727 non-null   object
 6   class     1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
df['class'].value_counts()
```

class	count
unacc	1209
acc	384
good	69
vgood	65

dtype: int64

```
X = df.drop(['class'], axis=1)

y = df['class']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=42)
```

```
X_train.shape, X_test.shape
```

```
((1157, 6), (570, 6))
```

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.3.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.1.4)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoc
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2024
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (20
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoder
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encoder
Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 2.0 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```
# Encode Categorical
import category_encoders as ce

# encode variables with ordinal encoding
encoder = ce.OrdinalEncoder(cols=['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety'])

X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)

X_train.head()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-3-3e45ef93a379> in <cell line: 2>()
      1 # Encode Categorical
----> 2 import category_encoders as ce
      3 
      4 # encode variables with ordinal encoding
      5 encoder = ce.OrdinalEncoder(cols=['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety'])

ModuleNotFoundError: No module named 'category_encoders'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

Next steps: Explain error

```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

```
y_pred_test = logreg.predict(X_test)
```

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.7702
```

```
y_pred_train = logreg.predict(X_train)

y_pred_train
```

```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.7891
```

```
# fit the Logsitic Regression model with C=100

# instantiate the model
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)


# fit the model
logreg100.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
```

```
Training set score: 0.7986
Test set score: 0.7754
```

```
from sklearn.model_selection import GridSearchCV
```

```
parameters = [{'C':[1, 10, 100, 1000]}]
```

```
grid_search = GridSearchCV(estimator = logreg,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)
```

```
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

```
# examine the best model

# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))

# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.best_params_))

# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.7952


Parameters that give the best results :

 {'C': 1000}


Estimator that was chosen by the search :

 LogisticRegression(C=1000, random_state=0, solver='liblinear')
```

```
# calculate GridSearch CV score on test set
```

```
print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
GridSearch CV score on test set: 0.7754
```

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

distributions = dict(C=uniform(loc=0, scale=4),
                     penalty=['l2', 'l1'])

randomized_search = RandomizedSearchCV(estimator = logreg,
                                param_distributions = distributions,
                                scoring = 'accuracy',
                                cv = 5,
                                verbose=0)
```

```
randomized_search.fit(X_train, y_train)
```

```
RandomizedSearchCV
estimator: LogisticRegression
LogisticRegression
```

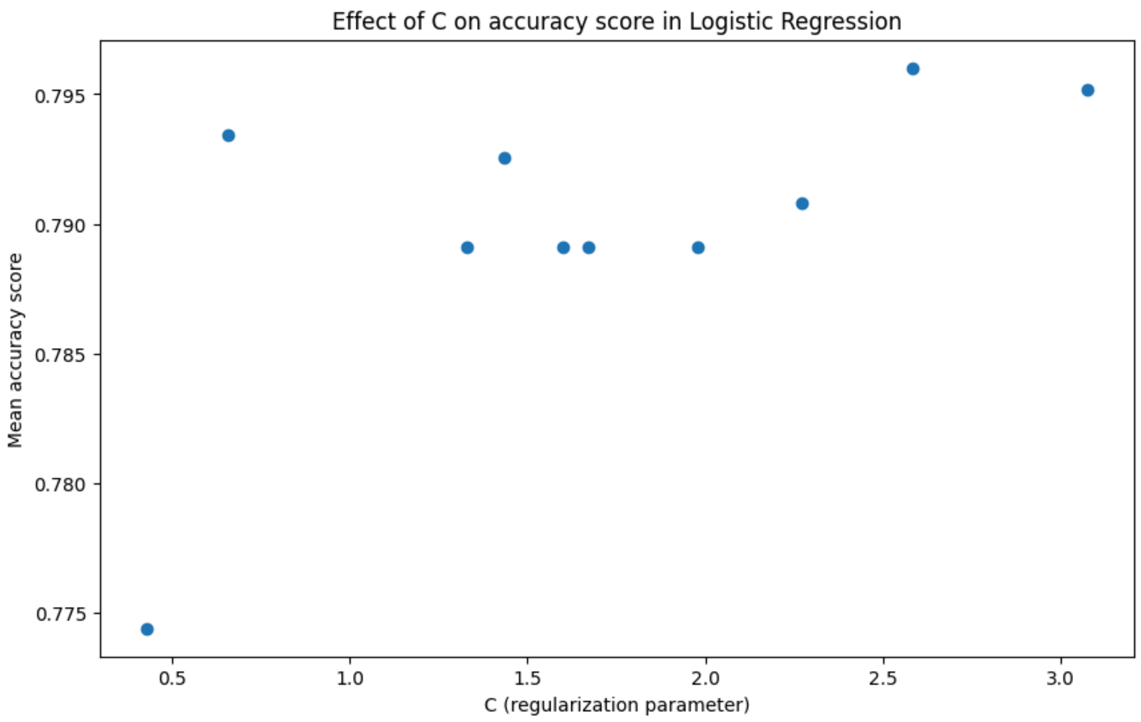
```
# examine the best model

# best score achieved during the GridSearchCV
print('RandomizedSearch CV best score : {:.4f}\n\n'.format(randomized_search.best_score_))

# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (randomized_search.best_params_))

# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (randomized_search.best_estimator_))
```

```
RandomizedSearch CV best score : 0.7960


Parameters that give the best results :

 {'C': 2.581310528951185, 'penalty': 'l1'}


Estimator that was chosen by the search :

 LogisticRegression(C=2.581310528951185, penalty='l1', random_state=0,
                   solver='liblinear')
```

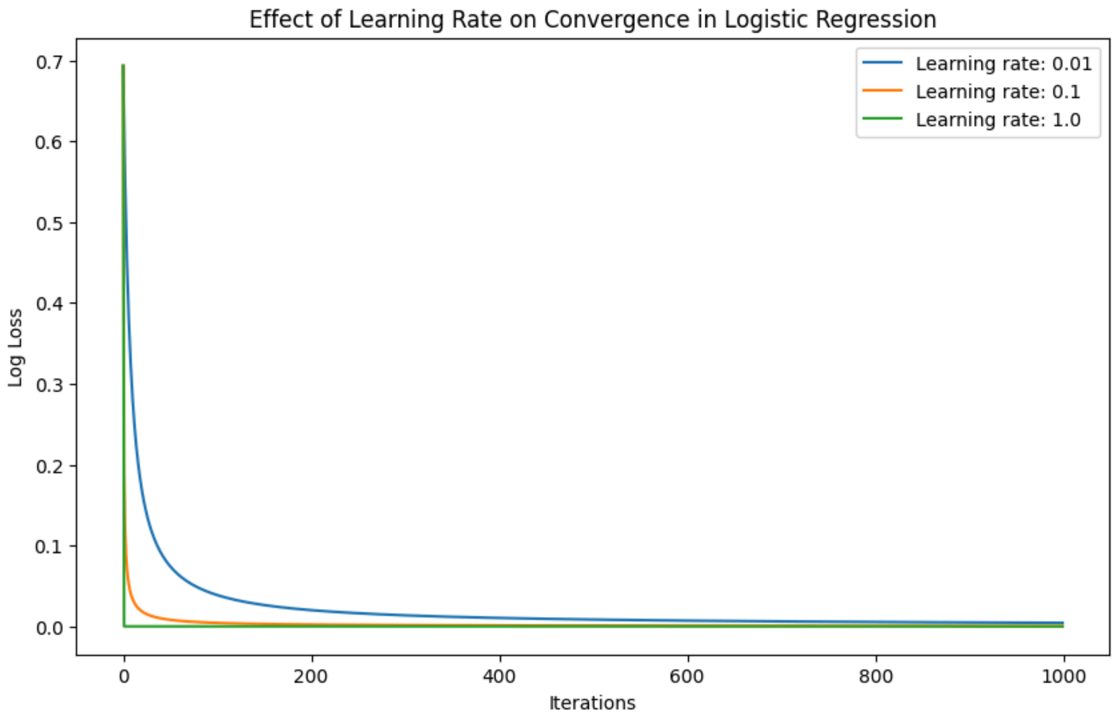
```
# calculate RandomizedSearch CV score on test set

print(' score on test set: {0:0.4f}'.format(randomized_search.score(X_test, y_test)))
```

```
 score on test set: 0.7719
```

```
# Plot the results
results = pd.DataFrame(randomized_search.cv_results_)
plt.figure(figsize=(10, 6))
plt.scatter(results['param_C'], results['mean_test_score'])
plt.xlabel('C (regularization parameter)')
plt.ylabel('Mean accuracy score')
plt.title('Effect of C on accuracy score in Logistic Regression')
plt.show()
```




```
#task2
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def log_loss(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

def logistic_regression_gd(X, y, learning_rate, num_iterations):
    m, n = X.shape
    theta = np.zeros(n)
    loss_history = []

    for _ in range(num_iterations):
        z = np.dot(X, theta)
        h = sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / m
        theta -= learning_rate * gradient
        loss_history.append(log_loss(y, h))

    return theta, loss_history

# Prepare data for binary classification (setosa vs. not setosa)
y_binary = (y_train == 0).astype(int)
X_train_with_bias = np.c_[np.ones((X_train.shape[0], 1)), X_train]

learning_rates = [0.01, 0.1, 1.0]
num_iterations = 1000

plt.figure(figsize=(10, 6))
for lr in learning_rates:
    theta, loss_history = logistic_regression_gd(X_train_with_bias, y_binary, lr, num_iterations)
    plt.plot(range(num_iterations), loss_history, label=f'Learning rate: {lr}')

plt.xlabel('Iterations')
plt.ylabel('Log Loss')
plt.title('Effect of Learning Rate on Convergence in Logistic Regression')
plt.legend()
plt.show()
```




```
#task3
regularizations = ['l1', 'l2', 'elasticnet', None]
C_values = [0.01, 0.1, 1, 10, 100]

results = []

for reg in regularizations:
    for C in C_values:
        if reg == 'elasticnet':
            model = LogisticRegression(penalty=reg, solver='saga', C=C, l1_ratio=0.5, random_state=42, max_iter=500)
        elif reg is None:
            model = LogisticRegression(penalty=reg, solver='lbfgs', C=C, random_state=42, max_iter=500)
        else:
            model = LogisticRegression(penalty=reg, solver='liblinear', C=C, random_state=42, max_iter=500)

        model.fit(X_train, y_train)
        train_score = model.score(X_train, y_train)
        test_score = model.score(X_test, y_test)
        results.append((reg, C, train_score, test_score))

results_df = pd.DataFrame(results, columns=['Regularization', 'C', 'Train Score', 'Test Score'])

plt.figure(figsize=(10, 6))
for reg in regularizations:
    reg_results = results_df[results_df['Regularization'] == reg]
    plt.plot(reg_results['C'], reg_results['Test Score'], marker='o', label=f'Regularization: {reg}')

plt.xscale('log')
plt.xlabel('C (inverse of regularization strength)')
plt.ylabel('Test Accuracy')
plt.title('Effect of Regularization and C on Logistic Regression Performance')
plt.legend()
plt.show()

print(results_df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
```